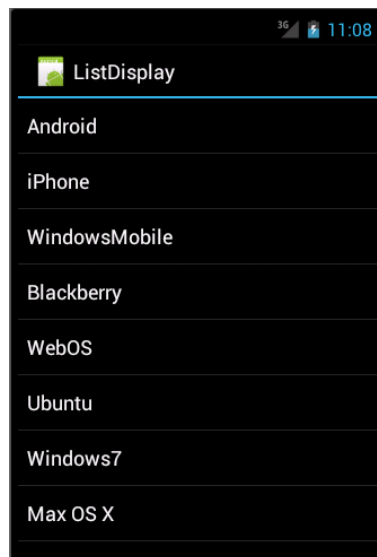


UNIT – 2 : Basic Attributes and Events of Important Android Widgets(UI)

1. List View:

Android **List View** is a view which groups several items and display them in vertical scrollable list. The list items are automatically inserted to the list using an **Adapter** that pulls content from a source such as an array or database



An adapter actually bridges between UI components and the data source that fill data into UI Component. Adapter holds the data and send the data to adapter view, the view can takes the data from adapter view and shows the data on different views like as spinner, list view, grid view etc.

The **List View** and **GridView** are subclasses of **AdapterView** and they can be populated by binding them to an **Adapter**, which retrieves data from an external source and creates a View that represents each data entry.

Android provides several subclasses of Adapter that are useful for retrieving different kinds of data and building views for an AdapterView (i.e. ListView or GridView). The common adapters are **ArrayAdapter**, **BaseAdapter**, **CursorAdapter**, **SimpleCursorAdapter**, **SpinnerAdapter** and **WrapperListAdapter**. We will see separate examples for both the adapters.

List View Attributes

Following are the important attributes specific to GridView –

Sr.No	Attribute & Description
1	android:id This is the ID which uniquely identifies the layout.
2	android:divider This is drawable or color to draw between list items.
3	android:dividerHeight This specifies height of the divider. This could be in px, dp, sp, in, or mm.
4	android:entries Specifies the reference to an array resource that will populate the ListView.
5	android:footerDividersEnabled When set to false, the ListView will not draw the divider before each footer view. The default value is true.
6	android:headerDividersEnabled When set to false, the ListView will not draw the divider after each header view. The default value is true.

ArrayAdapter

You can use this adapter when your data source is an array. By default, ArrayAdapter creates a view for each array item by calling `toString()` on each item and placing the contents in a **TextView**. Consider you have an array of strings you want to display in a **ListView**, initialize a new **ArrayAdapter** using a constructor to specify the layout for each string and the string array –

```
ArrayAdapter adapter = new ArrayAdapter<String>(this,R.layout.ListView,StringArray);
```

MOBILE APPLICATION DEVELOPMENT - 2

Here are arguments for this constructor –

- First argument **this** is the application context. Most of the case, keep it **this**.
- Second argument will be layout defined in XML file and having **TextView** for each string in the array.
- Final argument is an array of strings which will be populated in the text view.

Once you have array adapter created, then simply call **setAdapter()** on your **ListView** object as follows –

```
ListView listView = (ListView) findViewById(R.id.listview);  
listView.setAdapter(adapter);
```

You will define your list view under `res/layout` directory in an XML file. For our example we are going to use `activity_main.xml` file.

Example

Following is the example which will take you through simple steps to show how to create your own Android application using `ListView`. Follow the following steps to modify the Android application we created in *Hello World Example* chapter –

Step	Description
1	You will use Android Studio IDE to create an Android application and name it as <i>ListDisplay</i> under a package <i>com.example.ListDisplay</i> as explained in the <i>Hello World Example</i> chapter.
2	Modify the default content of <i>res/layout/activity_main.xml</i> file to include <code>ListView</code> content with the self explanatory attributes.
3	No need to change <i>string.xml</i> , Android studio takes care of default string constants.
4	Create a Text View file <i>res/layout/activity_listview.xml</i> . This file will have setting to display all the list items. So you can customize its fonts, padding, color etc. using this file.
6	Run the application to launch Android emulator and verify the result of the changes done in the application.

MOBILE APPLICATION DEVELOPMENT - 2

Following is the content of the modified main activity file **src/com.example.ListDisplay/ListDisplay.java**. This file can include each of the fundamental life cycle methods.

```
package com.example.ListDisplay;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class ListDisplay extends Activity {
    // Array of strings...
    String[] mobileArray = {"Android", "IPhone", "WindowsMobile", "Blackberry",
        "WebOS", "Ubuntu", "Windows7", "Max OS X"};

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        ArrayAdapter adapter = new ArrayAdapter<String>(this,
            R.layout.activity_listview, mobileArray);

        ListView listView = (ListView) findViewById(R.id.mobile_list);
        listView.setAdapter(adapter);
    }
}
```

Following will be the content of **res/layout/activity_main.xml** file –

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".ListActivity" >

    <ListView
        android:id="@+id/mobile_list"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
    </ListView>

</LinearLayout>
```

MOBILE APPLICATION DEVELOPMENT - 2


Following will be the content of **res/values/strings.xml** to define two new constants –

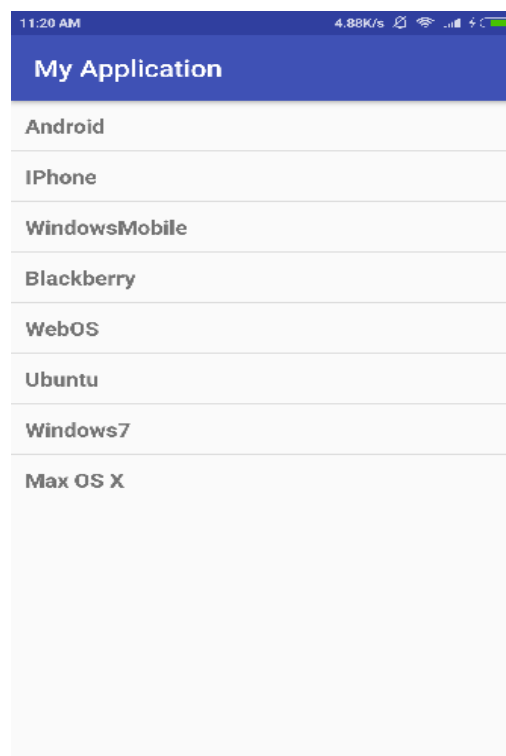
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">ListDisplay</string>
  <string name="action_settings">Settings</string>
</resources>
```

Following will be the content of **res/layout/activity_listview.xml** file –

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Single List Item Design -->

<TextView xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/label"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:padding="10dip"
  android:textSize="16dip"
  android:textStyle="bold" >
</TextView>
```

Let's try to run our modified **Hello World!** application we just modified. I assume you had created your **AVD** while doing environment set-up. To run the app from Android studio, open one of your project's activity files and click Run  icon from the tool bar. Android studio installs the app on your AVD and starts it and if everything is fine with your set-up and application, it will display following Emulator window –



SimpleCursorAdapter

You can use this adapter when your data source is a database Cursor. When using *SimpleCursorAdapter*, you must specify a layout to use for each row in the **Cursor** and which columns in the Cursor should be inserted into which views of the layout.

For example, if you want to create a list of people's names and phone numbers, you can perform a query that returns a Cursor containing a row for each person and columns for the names and numbers. You then create a string array specifying which columns from the Cursor you want in the layout for each result and an integer array specifying the corresponding views that each column should be placed –

```
String[] fromColumns = {ContactsContract.Data.DISPLAY_NAME,  
    ContactsContract.CommonDataKinds.Phone.NUMBER};  
int[] toViews = {R.id.display_name, R.id.phone_number};
```

When you instantiate the SimpleCursorAdapter, pass the layout to use for each result, the Cursor containing the results, and these two arrays –

```
SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,  
    R.layout.person_name_and_number, cursor, fromColumns, toViews, 0);  
  
ListView listView = getListView();  
listView.setAdapter(adapter);
```

The SimpleCursorAdapter then creates a view for each row in the Cursor using the provided layout by inserting each from Columns item into the corresponding **toViews** view.

2. Custom List View: (Adding Images, Sub-Titles):

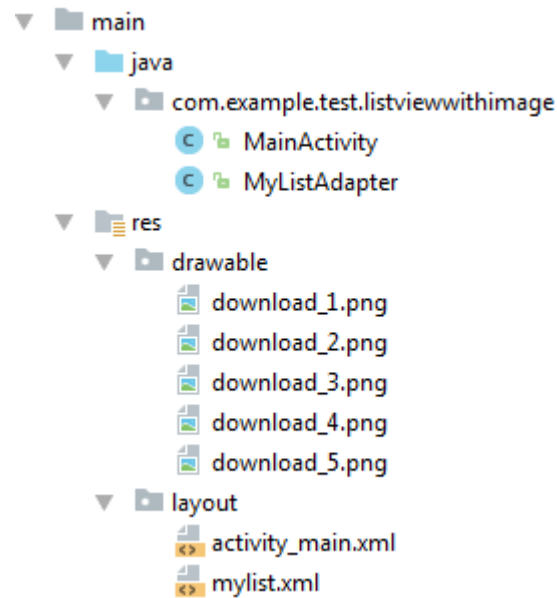
After creating simple ListView, android also provides facilities to customize our ListView.

As the simple ListView, custom ListView also uses Adapter classes which added the content from data source (such as string array, array, database etc). Adapter bridges data between an AdapterViews and other Views.

Example of Custom ListView

In this custom listview example, we are adding image, text with title and its sub-title.

Structure of custom listview project



activity_main.xml

Create an activity_main.xml file in layout folder.

File: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.test.listviewwithimage.MainActivity">

    <ListView
        android:id="@+id/list"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
        android:layout_marginBottom="50dp">
    </ListView>
</RelativeLayout>
```

Create an additional mylist.xml file in layout folder which contains view components displayed in listview.

mylist.xml

File: mylist.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >

    <ImageView
        android:id="@+id/icon"
        android:layout_width="60dp"
        android:layout_height="60dp"
        android:padding="5dp" />

    <LinearLayout android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <TextView
            android:id="@+id/title"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Medium Text"
            android:textStyle="bold"
            android:textAppearance="?android:attr/textAppearanceMedium"
            android:layout_marginLeft="10dp"
            android:layout_marginTop="5dp"
            android:padding="2dp"
            android:textColor="#4d4d4d" />

        <TextView
            android:id="@+id/subtitle"
```



```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="TextView"
        android:layout_marginLeft="10dp"/>
    </LinearLayout>
</LinearLayout>
```

Place the all required images in drawable folder.

Activity class

File: MainActivity.java

```
package com.example.test.listviewwithimage;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ListView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    ListView list;

    String[] maintitle = {
        "Title 1","Title 2",
        "Title 3","Title 4",
        "Title 5",
    };

    String[] subtitle = {
        "Sub Title 1","Sub Title 2",
        "Sub Title 3","Sub Title 4",
        "Sub Title 5",
    };

    Integer[] imgid={
        R.drawable.download_1,R.drawable.download_2,
```

MOBILE APPLICATION DEVELOPMENT - 2

```
R.drawable.download_3,R.drawable.download_4,  
R.drawable.download_5,  
};
```

```
@Override
```

```
protected void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);
```

```
    MyListAdapter adapter=new MyListAdapter(this, maintitle, subtitle,imgid);  
    list=(ListView)findViewById(R.id.list);  
    list.setAdapter(adapter);
```

```
    list.setOnItemClickListener(new AdapterView.OnItemClickListener() {
```

```
        @Override
```

```
        public void onItemClick(AdapterView<?> parent, View view,int position, long id) {
```

```
            // TODO Auto-generated method stub
```

```
            if(position == 0) {
```

```
                //code specific to first list item
```

```
                Toast.makeText(getApplicationContext(),"Place Your First Option Code",Toast.LENGTH_SHORT).show();  
            }
```

```
            else if(position == 1) {
```

```
                //code specific to 2nd list item
```

```
                Toast.makeText(getApplicationContext(),"Place Your Second Option Code",Toast.LENGTH_SHORT).show();  
            }
```

```
            else if(position == 2) {
```

```
                Toast.makeText(getApplicationContext(),"Place Your Third Option Code",Toast.LENGTH_SHORT).show();  
            }
```

```
            else if(position == 3) {
```

MOBILE APPLICATION DEVELOPMENT - 2

```
        Toast.makeText(getApplicationContext(),"Place Your Forth Option Code",Toast.LENGTH_SHORT).show();
    }
    else if(position == 4) {

        Toast.makeText(getApplicationContext(),"Place Your Fifth Option Code",Toast.LENGTH_SHORT).show();
    }

    }
    });
}
}
```

Customize Our ListView

Create another java class MyListView.java which extends ArrayAdapter class. This class customizes our listview.

MyListView.java

```
package com.example.test.listviewwithimage;

import android.app.Activity;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.TextView;

public class MyListAdapter extends ArrayAdapter<String> {

    private final Activity context;
    private final String[] maintitle;
    private final String[] subtitle;
    private final Integer[] imgid;
```

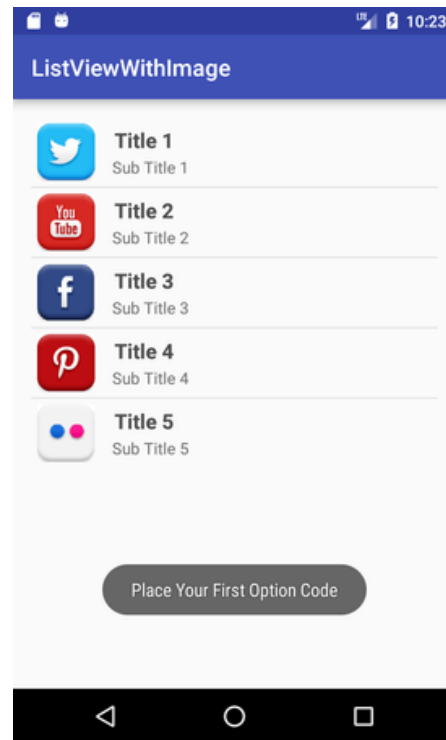
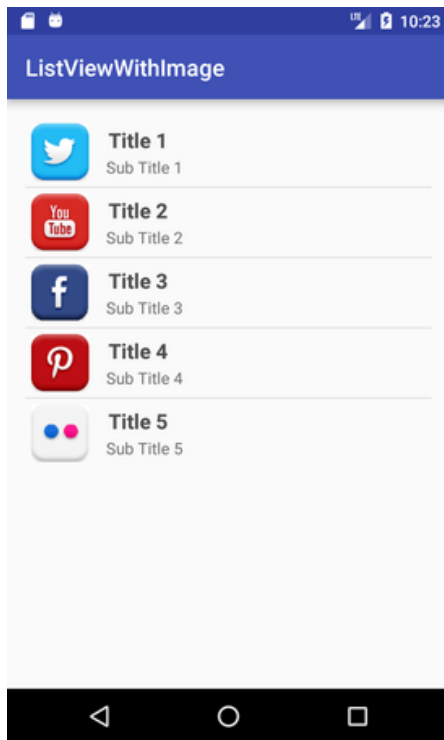
MOBILE APPLICATION DEVELOPMENT - 2

```
public MyListAdapter(Activity context, String[] maintitle,String[] subtitle, Integer[] imgid) {  
    super(context, R.layout.mylis, maintitle);  
    // TODO Auto-generated constructor stub  
  
    this.context=context;  
    this.maintitle=maintitle;  
    this.subtitle=subtitle;  
    this.imgid=imgid;  
  
}
```

```
public View getView(int position,View view,ViewGroup parent) {  
    LayoutInflater inflater=context.getLayoutInflater();  
    View rowView=inflater.inflate(R.layout.mylis, null,true);  
  
    TextView titleText = (TextView) rowView.findViewById(R.id.title);  
    ImageView imageView = (ImageView) rowView.findViewById(R.id.icon);  
    TextView subtitleText = (TextView) rowView.findViewById(R.id.subtitle);  
  
    titleText.setText(maintitle[position]);  
    imageView.setImageResource(imgid[position]);  
    subtitleText.setText(subtitle[position]);  
  
    return rowView;  
  
};  
}
```

Output

MOBILE APPLICATION DEVELOPMENT - 2



3. Date Picker:

Android Date Picker allows you to select the date consisting of day, month and year in your custom user interface. For this functionality android provides DatePicker and DatePickerDialog components.

Android DatePicker is a widget to select date. It allows you to select date by day, month and year. Like DatePicker, android also provides TimePicker to select time.

The android.widget.DatePicker is the subclass of FrameLayout class.

Android DatePicker Example

Let's see the simple example of datepicker widget in android.

activity_main.xml

File: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="example.javatpoint.com.datepicker.MainActivity">
```

<TextView

```
android:id="@+id/textView1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_above="@+id/button1"
android:layout_alignParentLeft="true"
android:layout_alignParentStart="true"
android:layout_marginBottom="102dp"
android:layout_marginLeft="30dp"
android:layout_marginStart="30dp"
android:text="" />
```

<Button

```
android:id="@+id/button1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignParentBottom="true"
android:layout_centerHorizontal="true"
android:layout_marginBottom="20dp"
android:text="Change Date" />
```

<DatePicker

```
android:id="@+id/datePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_above="@+id/textView1"
android:layout_centerHorizontal="true"
android:layout_marginBottom="36dp" />
```

</RelativeLayout>

Activity class

File: MainActivity.java

```
package example.javatpoint.com.datepicker;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
```

MOBILE APPLICATION DEVELOPMENT - 2

```
import android.view.View;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
    DatePicker picker;
    Button displayDate;
    TextView textview1;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        textview1=(TextView)findViewById(R.id.textView1);
        picker=(DatePicker)findViewById(R.id.datePicker);
        displayDate=(Button)findViewById(R.id.button1);

        textview1.setText("Current Date: "+getCurrentDate());

        displayDate.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View view) {

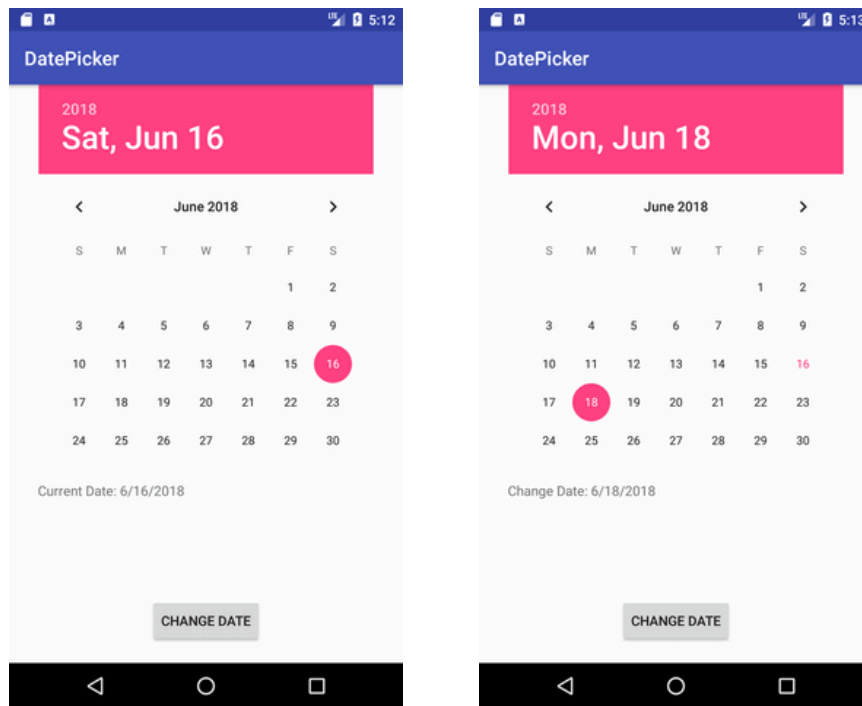
                textview1.setText("Change Date: "+getCurrentDate());
            }

        });

    }
    public String getCurrentDate(){
        StringBuilder builder=new StringBuilder();
        builder.append((picker.getMonth() + 1)+"/");//month is 0 based
        builder.append(picker.getDayOfMonth()+"/");
        builder.append(picker.getYear());
        return builder.toString();
    }
}
```

MOBILE APPLICATION DEVELOPMENT - 2

Output:



4. Time Picker:

Android Time Picker allows you to select the time of day in either 24 hour or AM/PM mode. The time consists of hours, minutes and clock format. Android provides this functionality through TimePicker class.

Android TimePicker widget is used to select date. It allows you to select time by hour and minute. You cannot select time by seconds.

The `android.widget.TimePicker` is the subclass of `FrameLayout` class.

Android TimePicker Example

Let's see a simple example of android time picker.

activity_main.xml

File: `activity_main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
```


MOBILE APPLICATION DEVELOPMENT - 2

```
android:layout_height="match_parent"  
tools:context="example.javatpoint.com.timepicker.MainActivity">
```

<TextView

```
    android:id="@+id/textView1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_above="@+id/button1"  
    android:layout_alignParentLeft="true"  
    android:layout_alignParentStart="true"  
    android:layout_marginBottom="102dp"  
    android:layout_marginLeft="30dp"  
    android:layout_marginStart="30dp"  
    android:text="" />
```

<Button

```
    android:id="@+id/button1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignParentBottom="true"  
    android:layout_centerHorizontal="true"  
    android:layout_marginBottom="20dp"  
    android:text="Change Time" />
```

<TimePicker

```
    android:id="@+id/timePicker"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_above="@+id/textView1"  
    android:layout_centerHorizontal="true"  
    android:layout_marginBottom="36dp" />
```

</RelativeLayout>

Activity class

File: MainActivity.java

```
package example.javatpoint.com.timepicker;
```

```
import android.support.v7.app.AppCompatActivity;
```

MOBILE APPLICATION DEVELOPMENT - 2

```
import android.os.Bundle;
```

```
import android.view.View;
```

```
import android.widget.Button;
```

```
import android.widget.TextView;
```

```
import android.widget.TimePicker;
```

```
public class MainActivity extends AppCompatActivity {
```

```
    TextView textview1;
```

```
    TimePicker timepicker;
```

```
    Button changetime;
```

```
@Override
```

```
protected void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);
```

```
    setContentView(R.layout.activity_main);
```

```
    textview1=(TextView)findViewById(R.id.textView1);
```

```
    timepicker=(TimePicker)findViewById(R.id.timePicker);
```

```
    //Uncomment the below line of code for 24 hour view
```

```
    timepicker.setIs24HourView(true);
```

```
    changetime=(Button)findViewById(R.id.button1);
```

```
    textview1.setText(getCurrentTime());
```

```
    changetime.setOnClickListener(new View.OnClickListener(){
```

```
        @Override
```

```
        public void onClick(View view) {
```

```
            textview1.setText(getCurrentTime());
```

```
        }
```

```
    });
```

```
}
```

```
public String getCurrentTime(){
```

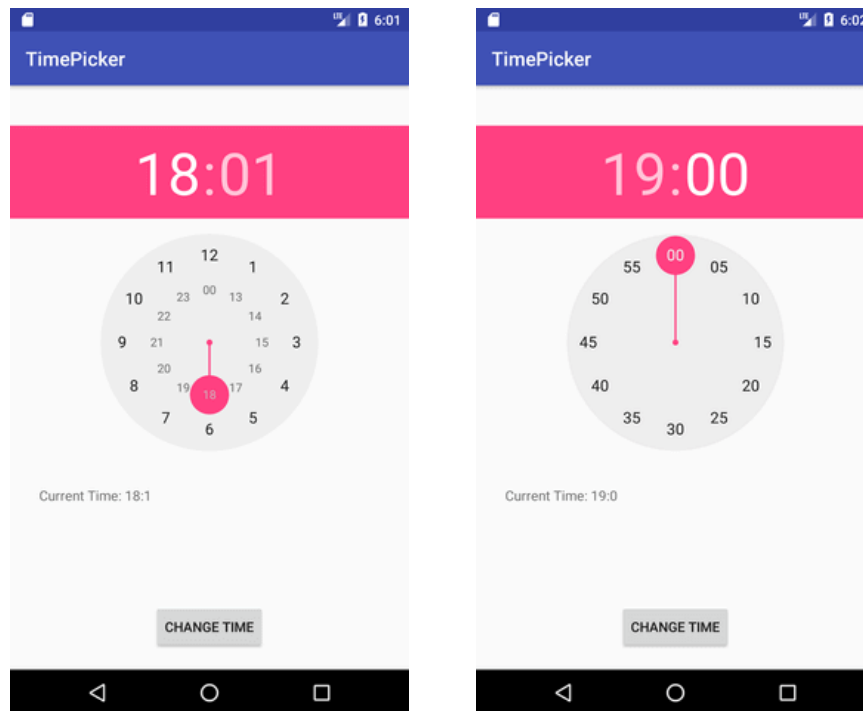
```
    String currentTime="Current Time: "+timepicker.getCurrentHour()+":"+timepicker.getCurrentMinute();
```

```
    return currentTime;
```

```
}
```

```
}
```

Output:



5. ProgressBar:

Progress bars are used to show progress of a task. For example, when you are uploading or downloading something from the internet, it is better to show the progress of download/upload to the user.

In android there is a class called `ProgressDialog` that allows you to create progress bar. In order to do this, you need to instantiate an object of this class. Its syntax is.

```
ProgressDialog progress = new ProgressDialog(this);
```

We can display the **android progress bar** dialog box to display the status of work being done e.g. downloading file, analyzing status of work etc.

In this example, we are displaying the progress dialog for dummy file download operation.

Here we are using **android.app.ProgressDialog** class to show the progress bar. Android `ProgressDialog` is the subclass of `AlertDialog` class.

The **ProgressDialog** class provides methods to work on progress bar like `setProgress()`, `setMessage()`, `setProgressStyle()`, `setMax()`, `show()` etc. The progress range of Progress Dialog is 0 to 10000.

Let's see a simple example to display progress bar in android.

```
ProgressDialog progressBar = new ProgressDialog(this);
progressBar.setCancelable(true);//you can cancel it by pressing back button
progressBar.setMessage("File downloading ...");
progressBar.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
progressBar.setProgress(0);//initially progress is 0
progressBar.setMax(100);//sets the maximum value 100
progressBar.show();//displays the progress bar
```

Android Progress Bar Example by ProgressDialog

Let's see a simple example to create progress bar using ProgressDialog class.

activity_main.xml

Drag one button from the palette, now the activity_main.xml file will look like this:

File: activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >
```

<Button

```
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="116dp"
    android:text="download file" />
```

```
</RelativeLayout>
```

MOBILE APPLICATION DEVELOPMENT - 2

Activity class

Let's write the code to display the progress bar dialog box.

File: MainActivity.java

```
package example.javatpoint.com.progressbar;

import android.app.ProgressDialog;
import android.os.Handler;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {
    Button btnStartProgress;
    ProgressDialog progressBar;
    private int progressBarStatus = 0;
    private Handler progressBarHandler = new Handler();
    private long fileSize = 0;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        addListenerOnButtonClick();
    }
    public void addListenerOnButtonClick() {
        btnStartProgress = findViewById(R.id.button);
        btnStartProgress.setOnClickListener(new View.OnClickListener(){

            @Override
            public void onClick(View v) {
                // creating progress bar dialog
                progressBar = new ProgressDialog(v.getContext());
                progressBar.setCancelable(true);
                progressBar.setMessage("File downloading ...");
                progressBar.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
                progressBar.setProgress(0);
                progressBar.setMax(100);
```

MOBILE APPLICATION DEVELOPMENT - 2

```
        progressBar.show();
        //reset progress bar and filesize status
        progressBarStatus = 0;
        fileSize = 0;

        new Thread(new Runnable() {
            public void run() {
                while (progressBarStatus < 100) {
                    // performing operation
                    progressBarStatus = doOperation();
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    // Updating the progress bar
                    progressBarHandler.post(new Runnable() {
                        public void run() {
                            progressBar.setProgress(progressBarStatus);
                        }
                    });
                }
                // performing operation if file is downloaded,
                if (progressBarStatus >= 100) {
                    // sleeping for 1 second after operation completed
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    // close the progress bar dialog
                    progressBar.dismiss();
                }
            }
        }).start();
    } //end of onClick method
});
}
```

MOBILE APPLICATION DEVELOPMENT - 2

// checking how much file is downloaded and updating the filesize

```
public int doOperation() {  
    //The range of ProgressDialog starts from 0 to 10000  
    while (fileSize <= 10000) {  
        fileSize++;  
        if (fileSize == 1000) {  
            return 10;  
        } else if (fileSize == 2000) {  
            return 20;  
        } else if (fileSize == 3000) {  
            return 30;  
        } else if (fileSize == 4000) {  
            return 40; // you can add more else if  
        }  
        /* else {  
            return 100;  
        }*/  
    } //end of while  
    return 100;  
} //end of doOperation
```

Output:

